This Page Is Inserted by IFW Operations
and is not a part of the Official Record

# BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

**As rescanning documents *will not* correct images, please do not report the images to the Image Problems Mailbox.**

OPIC
OFFICE DE LA PROPRIÉTÉ INTELLECTUELLE DU CANADA

CIPO
CANADIAN INTELLECTUAL PROPERTY OFFICE

(12) (19) (CA) **Demande-Application**

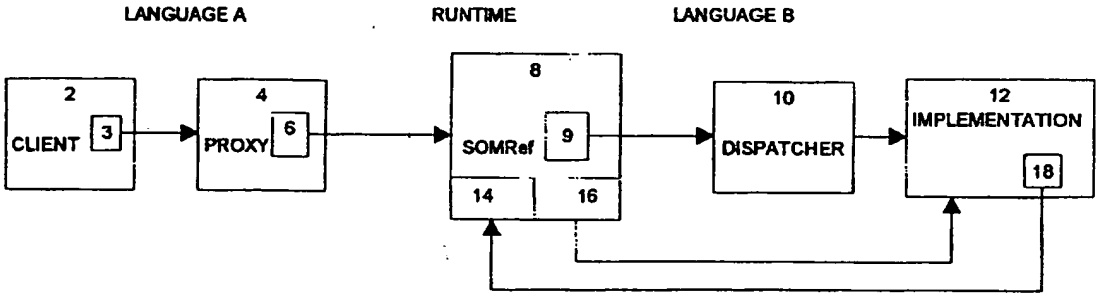
(21) (A1) **2,204,974**
(22) 1997/05/09
(43) 1998/11/09

(72) CHENG, Michael, US
(72) HANSON, Tim S., US
(72) NASH, Simon Christopher, US
(72) RAPER, Larry Keith, US
(72) ROCHAT, Kim Lawson, US
(72) THOMSON, Brian Ward, CA
(71) IBM CANADA LIMITED - IBM CANADA LIMITEE, CA

(51) Int.Cl.[6] G06F 9/44

(54) **UTILISATION TRANSPARENTE D'OBJETS COMPILES OU INTERPRETES DANS UN SYSTEME ORIENTE OBJETS**

(54) **TRANSPARENT USE OF COMPILED OR INTERPRETED OBJECTS IN AN OBJECT ORIENTED SYSTEM**




(57) L'invention est une méthode de prise en charge de mandataires en langage croisé. Une composante opérationnelle conserve un jeton interne indépendant du langage qui correspond à chaque objet exposé à une frontière linguistique. Chaque objet exposé est représenté par un objet mandataire dans la langue étrangère. Chaque fois qu'un renvoi à un objet traverse une frontière linguistique, c'est le jeton qui effectue cette traversée. Si le renvoi à transmettre est un objet mandataire, le jeton est extrait du mandataire et effectue la traversée.

(57) Support for cross-language proxies is provided. A runtime component maintains an internal, language-independent token corresponding to each object exposed across a language boundary. Each exposed object is represented in the foreign language by a proxy object. Whenever a reference to an object is passed across a language boundary, it is the token that is passed. If the reference to be passed is a proxy object, then the token is retrieved from the proxy and passed.

CA9-97-014

# TRANSPARENT USE OF COMPILED OR INTERPRETED OBJECTS IN AN OBJECT ORIENTED SYSTEM

## ABSTRACT

Support for cross-language proxies is provided. A runtime component maintains an internal, language-independent token corresponding to each object exposed across a language boundary. Each exposed object is represented in the foreign language by a proxy object. Whenever a reference to an object is passed across a language boundary, it is the token that is passed. If the reference to be passed is a proxy object, then the token is retrieved from the proxy and passed.

CA9-97-014

# TRANSPARENT USE OF COMPILED OR INTERPRETED OBJECTS IN AN OBJECT ORIENTED SYSTEM

## Field of the Invention

The present invention relates, in general, to the field of mixed object oriented language programming environments. The invention provides a mechanism to support cross-language proxies in such an environment where the requester (client) does not know what the target language is.

## Background of the Invention

Concurrently-filed application titled "Uniform Access To And Interchange Between Objects Employing A Plurality Of Access Methods" (IBM Docket No. CA997013), which is commonly assigned, describes in some detail the general nature of object oriented programming languages.

Briefly, object oriented (OO) technology is based on the notions of "object" and "class inheritance". An object is a reusable package of data that is a member of a class. The class provides the definition for a set of operations that can be performed on the object and other objects that are members of the same class. Classes can be related to one other by inheritance, and the properties, behaviours, data and operations of a parent or "base" class (a class without member objects), may be inherited with or without modification in a derived class under the control of a programmer.

OO programming languages, such as C++ and OO COBOL, are compiled languages. In C++, for example, the compiler defines an object data structure for each class. This structure includes the layout of the data members and, where appropriate, a virtual function table to implement indirect calls.

Other OO programming languages, such as Java and Smalltalk are interpreted languages. These

typically use a virtual machine to process source code or partially pre-compiled source code to calls to base functions or "primitives" resident in the virtual machine.

The notion of "proxies" assisting in providing local transparency for access to remote objects is discussed in the above referenced concurrently-filed application (IBM Docket No. CA997013). Proxies allow the uniform invocation of methods for same-language support, irrespective of the location of the objects. However, proxies alone are not sufficient to provide location transparency in the presence of cross-language calls.

## Summary of the Invention

It is an object of the present invention to provide a mechanism for transparent access to compiled or interpreted objects, irrespective of the nature of the object oriented programming environment from which the call is initiated.

It is also an object of the present invention to provide transparent access from a C++ environment to objects implemented variously in Java, OO COBOL and Smalltalk.

It is also an object of the present invention to provide access from program written in a "client" language or a combination of several client languages, to objects resident in the same process and implemented in an "implementation" language. The client and implementation languages can include C++, Java, COBOL, Smalltalk and other applicable OO languages.

It is also an object of the present invention to enable the roles of "client" and "implementation" be freely intermixed, such that any implementation of a first object in a first programming language may concurrently function as a client of a second object implemented in a second language, whether the second language is identical to or distinct from the first language.

It is also an object of the present invention that the access provided from a client language to an implementation language shall be "transparent", with the meaning that the construction of the program functioning in a client role shall not depend on the identity of the implementation language.

Accordingly, the present invention provides a method for effecting an access request from a client in a first language environment to an implementation object in a second language environment. The method includes the steps of converting the access request into a language-independent token, passing the token to the second language environment, and converting the token into an access request having parameters appropriate to the second language environment.

The invention also provides a mechanism for effecting cross-language access requests in an computing environment. This mechanism consists of a runtime component having means for generating language independent tokens representing implementation object, means for caching generating tokens, including reference means for locating cached tokens representing specific implementation objects, and means for passing access requests in a first language on implementation objects in a second language through tokens representing said implementation objects.

Computer program products embodying the above method steps are included as a non-obvious combination.

## Brief Description of the Drawings

Embodiments of the invention will be described in detail in association with the accompanying drawings in which:

Figures 1 to 3 are schematic diagrams illustrating the interfaces between clients and cross-language objects, according to aspects of the invention; and

Figures 4 to 6 are flow diagrams illustrating the steps to issue a cross-language call under the schemes illustrated in Figures 1 to 3.

CA9-97-014

Detailed Description of the Preferred Embodiments

The three principal elements of the invention are the proxies provided in the client language or languages, a dispatcher provided in the implementation language, and a runtime component which supports the generation, maintenance, and manipulation of language-independent object tokens each of which corresponds to an object made available for access from other languages.

In the preferred embodiment of the invention, the proxies provide client access to an object in accordance with the standards and practices defined by the Object Management Group's Common Object Request Broker Architecture (OMG CORBA). These standards specify an Interface Definition Language (IDL), a set of data types usable from IDL, and how these data types and other IDL artifacts are to be represented in different programming languages.

In the preferred embodiment, the proxies and dispatcher convert data types and object accesses between the language-specific forms mandated by the OMG CORBA standards and language-independent forms. The language-independent object token, a "SOMRef" in the preferred embodiment, is one of these forms, and is created, destroyed, invoked, and manipulated by the proxies and dispatcher using the support provided by the runtime component.

The above described technology provides a solution for mixing Java and C++, and is extensible to include at least Smalltalk and OO COBOL.

Figure 1 schematically illustrates a cross-language method call from client language A to an implementation language B. Figure 4 is a flow diagram illustrating the sequence of steps taken to implement this call.

Referring generally to Figure 1, the client 2 is presumed to have acquired, through means described

below, a reference 3 to a proxy object 4. As shown in block 100 of Figure 4, the client 2 uses reference 3 to invoke a method on the proxy object 4, which supports a method corresponding to each method supported by the target implementation 12.

The method of the proxy object 4 converts all parameters into their language-independent forms (block 102 of Figure 4). The proxy also determines the identity of the SOMRef 8 that represents the implementation (block 104 of Figure 4). This is accomplished either by consulting a pointer 6 to the SOMRef stored in the instance data of the proxy 4, or by consulting a separate lookup table (not shown). The proxy object's method then invokes an entry point in the runtime, passing the pointer to the target SOMRef 8, the language-independent forms of the parameters, and an identifier for the method to be called (block 106 of Figure 4).

The runtime component stores in each SOMRef 8 a pointer or collection of pointers 9 to a dispatcher 10. Dispatchers may be shared by multiple instances of an implementation, and in that case the SOMRef also stores a pointer 16 to the implementation itself. The dispatcher for a compiled language consists of a collection of functions, one for each method supported by the implementation. The implementation passes the address of each such function to the runtime before the first implementation object 12 is created, and the runtime saves these pointers in its collection of pointers 9. Using the method identifier given it by the proxy, the runtime selects the corresponding dispatch function pointer from its collection of pointers and calls it with the value of the pointer 16 to the implementation and the language-independent forms of the parameters (block 108 of Figure 4).

The dispatcher function converts the parameters from their language-independent forms into the language-dependent forms appropriate to language B (block 110 of Figure 4). It then invokes the method on the implementation 12, converts any return value or exception into a language-independent form, and returns it back through the runtime to the client 2 (blocks 112, 114 and 116 of Figure 4).

Dispatchers for interpreted languages such as Java and Smalltalk are objects with dispatcher methods rather than collections of dispatcher functions. Instead of selecting a function pointer from a collection of such pointers, the runtime uses the existing invocation capabilities of the Java or Smalltalk interpreter to invoke an appropriate dispatch method on the dispatcher object. The subsequent operation of the dispatch method is as described for the compiled language dispatch functions.

Figure 2 illustrates a more complex example in which object references are passed as parameters. Figure 5 is the corresponding flow diagram illustrating the sequence of steps.

Referring generally to Figure 2, client 22 has a pointer 23 to the proxy 24 for the target object implemented in language B. The client also has pointers 20 to an object implemented in language A, and 21 to a second proxy for a second object implemented in language B.

As before, client 22 calls a method on proxy 24 (block 120 of Figure 5), but this time included in the parameters of the call are the pointer values 20 and 21. Parameters are again converted by the proxy into a language-independent form (block 122 of Figure 5). The language-independent form of an object reference is the SOMRef, and proxy 24 converts pointer 21 into its language-independent form by finding the SOMRef 48 associated with the proxy 44.

In the preferred embodiment, all proxies inherit a common base class and the SOMRef pointer 46 is stored as instance data introduced by the common base class. Proxy 24 can convert the parameter pointer into a SOMRef by accessing that instance data, either directly or through an accessor method also introduced by the common proxy base class, whichever is more appropriate for the language A.

Other means including lookup tables can be used to store the correspondence between proxies and SOMRefs in embodiments that do not utilize a common base class.

The proxy also converts pointer 20 into a SOMRef (block 126 of Figure 5). In the preferred embodiment, implementations also inherit the common base class and therefore proxy 24 can access the SOMRef pointer 78 in the implementation 72 just as it was able to do so in the proxy 44.

However, to reduce the time and space associated with SOMRef construction SOMRefs are not created for implementations until they are required, and therefore, pointer 78 may be null if the implementation 72 has not previously been used in a cross-language call. If pointer 78 is null, proxy 24 will use the runtime facilities to allocate a SOMRef 68 for the implementation and will store the SOMRef's address into pointer 78 for future reuse (blocks 124, 128 of Figure 5).

The method call proceeds through the runtime as before and a dispatch method or function in the dispatcher 30 is invoked (blocks 130, 132 of Figure 5). Among the language-independent form parameters it receives are the SOMRefs 48 and 68, which it must convert into forms appropriate for the language B.

First the dispatcher invokes a function in the runtime to test whether the object behind the SOMRef 48 is implemented in language B (block 134 of Figure 5). This runtime function takes as arguments the SOMRef pointer, an integer value to identify language B drawn from a roster of language identifiers created for this purpose, and the name of the desired interface on the target object. The runtime passes this request on to dispatcher 50, which finds that the language identifier does match the implementation language B. Dispatcher 50 then returns a native language reference to the named interface of implementation object 52, through the runtime back to dispatcher 30, which then passes this native reference on its call to implementation object 32 (blocks 136, 138 of Figure 5). In this way implementation 32 receives a direct reference to same-language implementation 52 and can use it directly without further involvement of any proxies or of the runtime.

Dispatcher 30 also performs the same check for SOMRef 68, but when the language identifier is checked in dispatcher 70 it is found not to match language A. Dispatcher 30 then proceeds by calling

a second runtime function to query the SOMRef to determine whether an appropriate language B proxy already exists for it (blocks 136, 140 of Figure 5). The SOMRef maintains a collection 74 of pointers to proxies, indexed by language and interface identifiers. If an appropriate proxy is found it will be used in the call to implementation 32, otherwise dispatcher 30 allocates a new proxy 64 and associates it with SOMRef 68, including adding it to the collection 74 indexed by the correct language and interface identifiers (blocks 142, 144, 146 of Figure 5).

Returning of object references as return values is identical to the procedure described above except that the dispatcher object converts implementation and proxy references into SOMRefs, and the proxy object converts SOMRefs back into implementation and proxy references (blocks 148, 150, 152 of Figure 5).

Given one initial cross-language reference through a proxy, SOMRef, and dispatcher, the passing of object references as parameters and return values allows any number of other objects to be exposed between languages.

Figure 3 schematically illustrates how a client in language A receives an initial reference to a cross-language object without first possessing one. Figure 6 is the corresponding flow diagram.

Initially only the client 82 and proxy class 83 exist in language A (referring generally to Figure 3). The proxy class consists of proxy behaviours that are accessible by name without requiring the existence of a proxy instance. The precise form taken by these behaviors is language dependent: for C++ or Java they appear as C++ static methods, and in Smalltalk as class methods.

One of the proxy class behaviors is a "create" function. Client 82 calls the create function provided by proxy class 83 (block 160 of Figure 6). The create function calls a runtime entry point, passing it the name of the class for which an instance is to be created (block 162 of Figure 6). The runtime maintains a table of implementations 87 indexed by class name, and looks up the desired class in that

CA9-97-014

The embodiments of the invention in which an exclusive property or privilege is claimed are defined as follows:

1. A method for effecting an access request from a client in a first language environment to an implementation object in a second language environment, comprising:

converting the access request into a language-independent token;

passing the token to the second language environment; and

converting the token into an access request having parameters appropriate to the second language environment.

2. A method, according to claim 1, wherein the step of converting the access request into a language independent token comprises:

generating a proxy having methods corresponding to the implementation object; and

passing a reference from the client to the proxy invoking methods in said proxy to convert all parameters in the access request to a language independent form.

3. A method, according to claim 2, wherein the step of converting the access request into a language independent token further comprises:

identifying a token representing the implementation object; and

passing a pointer to the identified token.

4. A mechanism for effecting cross-language access requests in a computing environment, comprising:

a runtime component having means for generating language independent tokens representing implementation objects;

means for caching generated tokens, including reference means for locating cached tokens representing specific implementation objects; and

means for passing access requests in a first language on implementation objects in a second language through tokens representing said implementation objects.

5. A mechanism, according to claim 4, further comprising means for converting all parameters of an access request in the first language into a language-independent form and for passing a pointer from said language independent form to a token representing a target implementation object for said access request.

6. A mechanism, according to claim 6 wherein the runtime component further adapted to provide a dispatch function to convert the parameters of the access request to a language specific form appropriate to the target implementation object.

7. A mechanism, according to claim 6, wherein the dispatch function comprises, in the case of a compiled object oriented language:

a store of pointers to functions corresponding to each method supported by the target implementation; and

means to match a method provided by the proxy with a pointer to one of the functions.

8. A mechanism, according to claim 6, wherein the dispatch function comprises, in the case of an interpreted object oriented language:

objects having methods corresponding to each method supported by the target implementation object; and

means in the runtime to invoke an appropriate dispatch method for the target implementation object on a object.

9. A computer program product for effecting an access request from a client in a first language environment to an implementation object in a second language environment, said computer program product comprising:

a computer readable storage medium having computer readable code means embodied in said medium, said computer readable code means comprising:

computer instruction means for converting the access request into a language independent token;

computer instruction means for passing the token to the second language environment; and

computer instruction means for converting the token into an access request having parameters appropriate to the second language requirement.

FIGURE 1

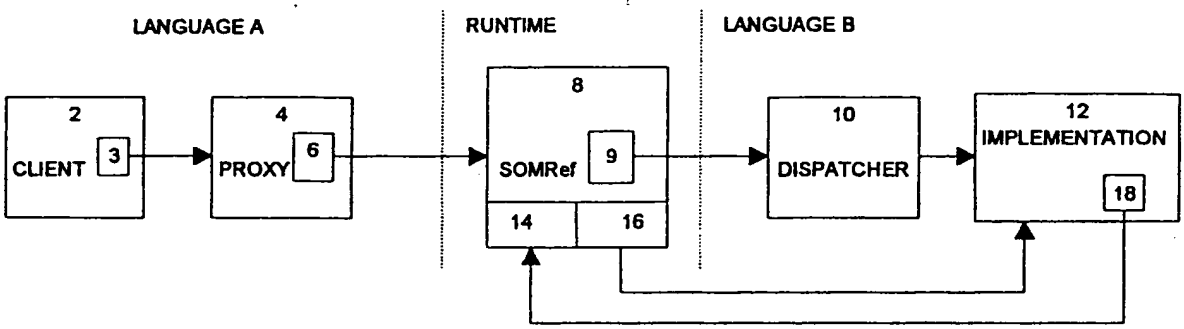

LANGUAGE A
RUNTIME
LANGUAGE B
2
CLIENT
3
4
PROXY
6
8
SOMRef
9
14
16
10
DISPATCHER
12
IMPLEMENTATION
18

FIGURE 2

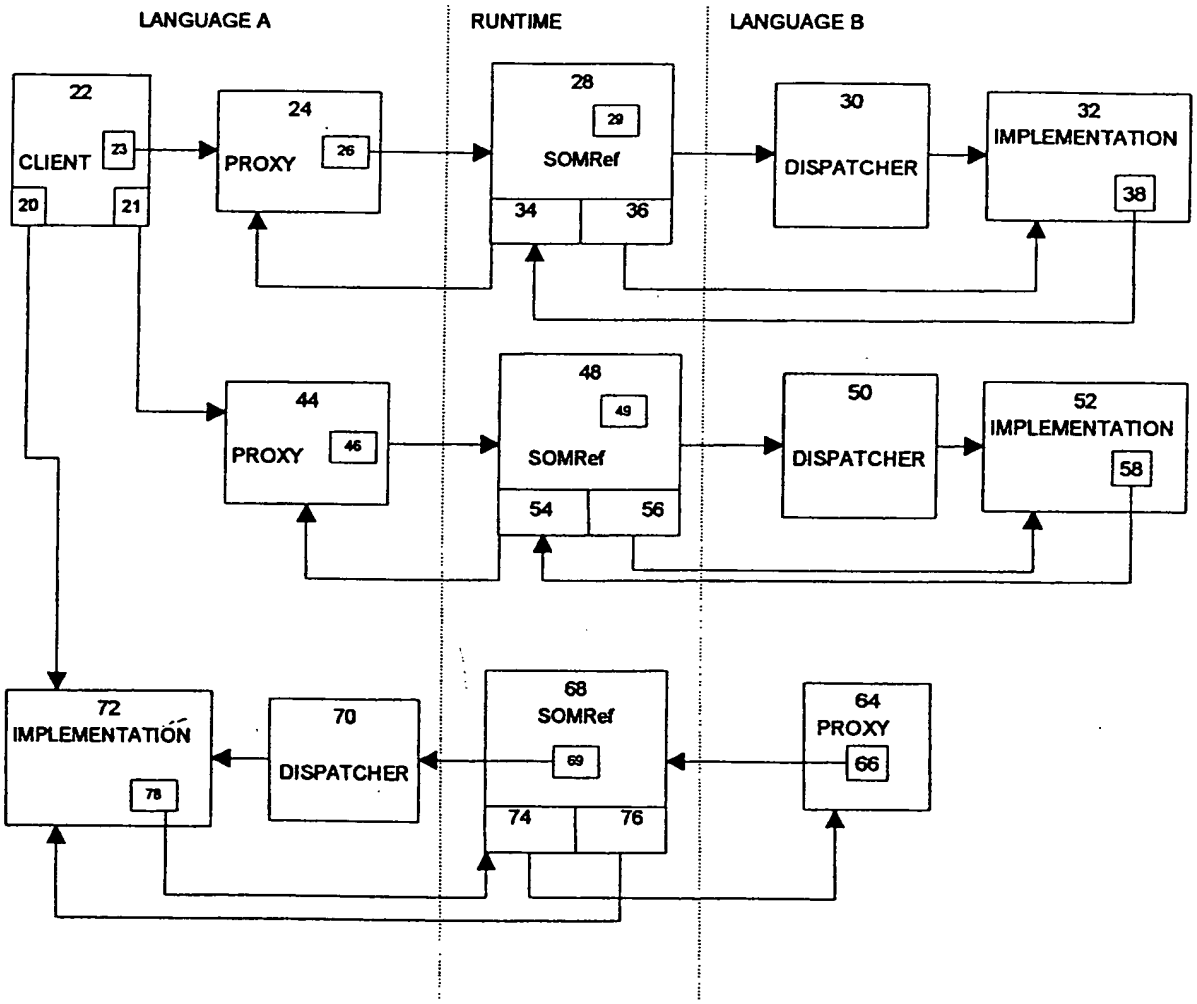
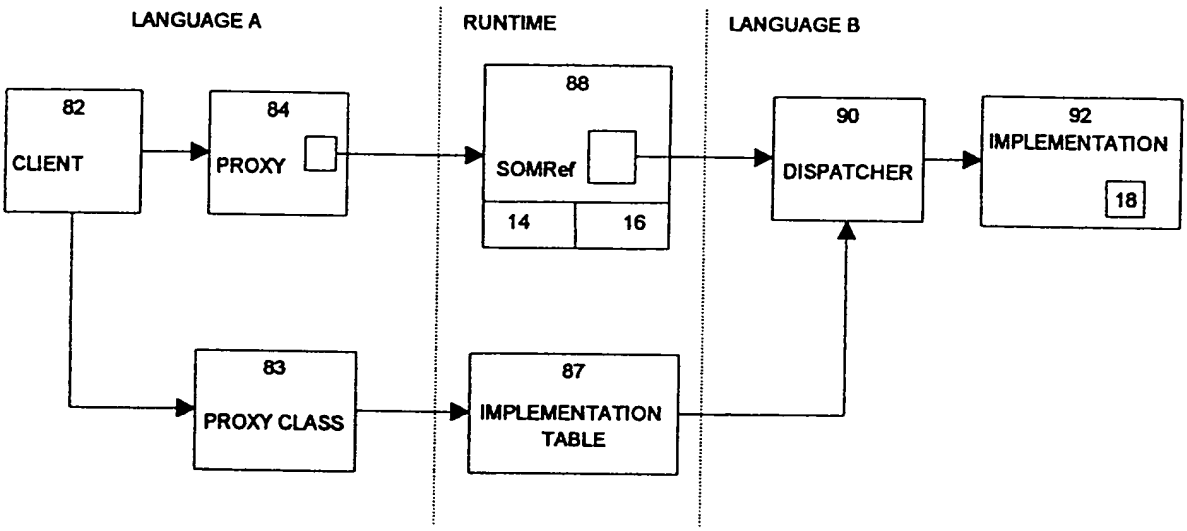

LANGUAGE A
RUNTIME
LANGUAGE B
22
CLIENT
23
20
21
24
PROXY
26
28
29
SOMRef
34
36
30
DISPATCHER
32
IMPLEMENTATION
38
44
PROXY
46
48
49
SOMRef
54
56
50
DISPATCHER
52
IMPLEMENTATION
58
72
IMPLEMENTATION
78
70
DISPATCHER
68
SOMRef
69
74
76
64
PROXY
66

FIGURE 3


LANGUAGE A
RUNTIME
LANGUAGE B
82
CLIENT
84
PROXY
88
SOMRef
14
16
90
DISPATCHER
92
IMPLEMENTATION
18
83
PROXY CLASS
87
IMPLEMENTATION TABLE

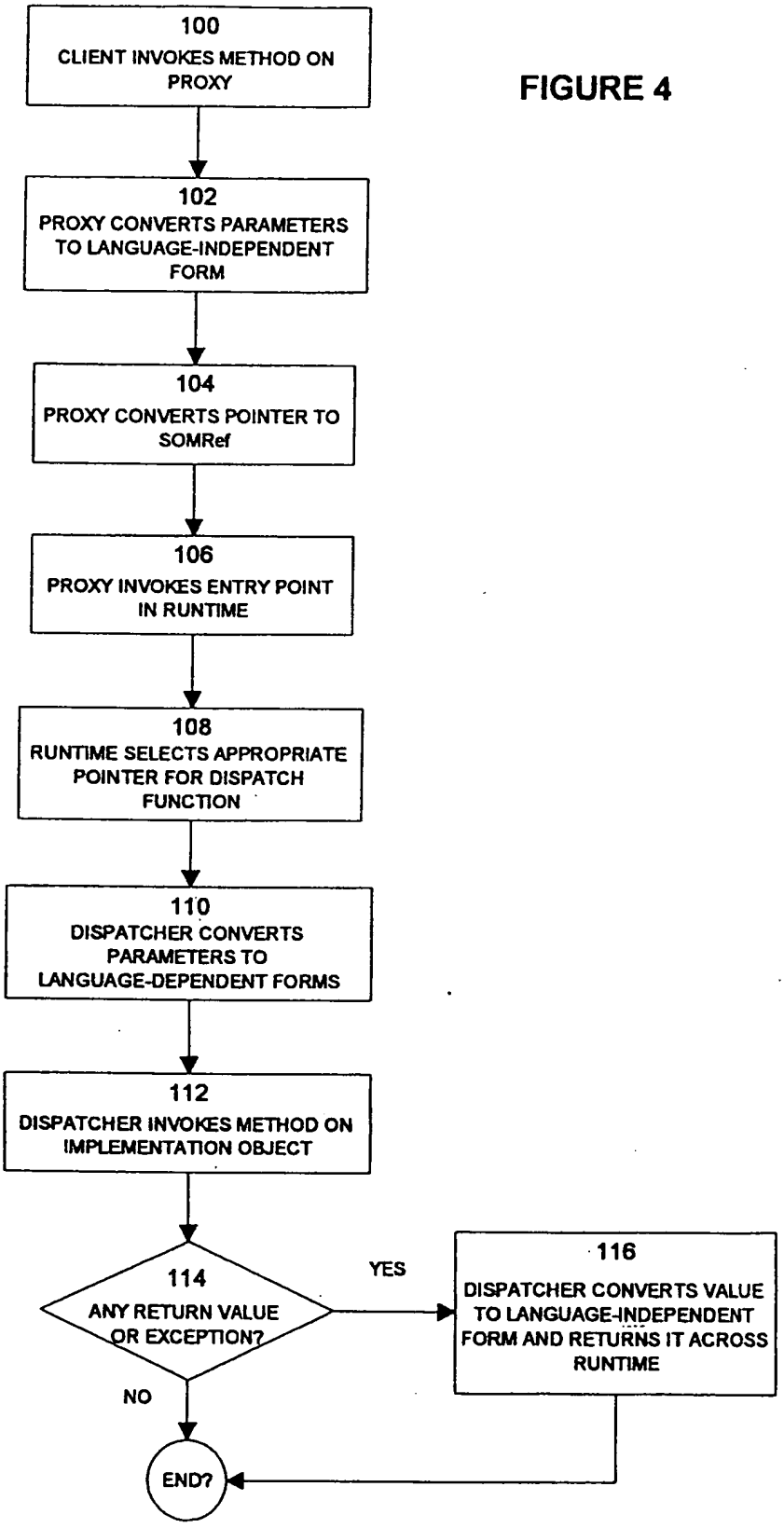
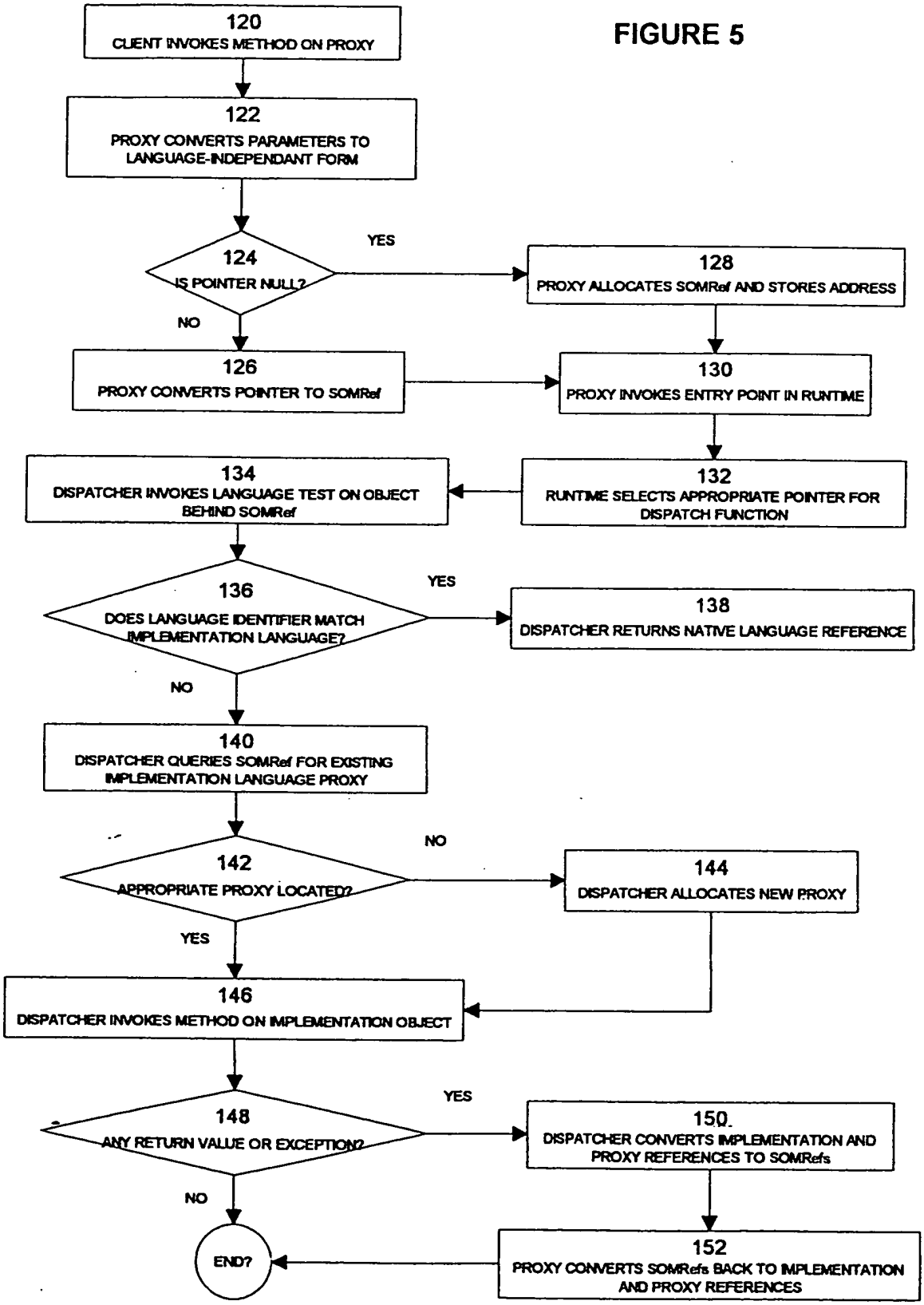

100
CLIENT INVOKES METHOD ON PROXY
102
PROXY CONVERTS PARAMETERS TO LANGUAGE-INDEPENDENT FORM
104
PROXY CONVERTS POINTER TO SOMRef
106
PROXY INVOKES ENTRY POINT IN RUNTIME
108
RUNTIME SELECTS APPROPRIATE POINTER FOR DISPATCH FUNCTION
110
DISPATCHER CONVERTS PARAMETERS TO LANGUAGE-DEPENDENT FORMS
112
DISPATCHER INVOKES METHOD ON IMPLEMENTATION OBJECT
114
ANY RETURN VALUE OR EXCEPTION?
YES
116
DISPATCHER CONVERTS VALUE TO LANGUAGE-INDEPENDENT FORM AND RETURNS IT ACROSS RUNTIME
NO
END?

FIGURE 4

FIGURE 5


120
CLIENT INVOKES METHOD ON PROXY
122
PROXY CONVERTS PARAMETERS TO LANGUAGE-INDEPENDANT FORM
124
IS POINTER NULL?
YES
128
PROXY ALLOCATES SOMRef AND STORES ADDRESS
NO
126
PROXY CONVERTS POINTER TO SOMRef
130
PROXY INVOKES ENTRY POINT IN RUNTIME
134
DISPATCHER INVOKES LANGUAGE TEST ON OBJECT BEHIND SOMRef
132
RUNTIME SELECTS APPROPRIATE POINTER FOR DISPATCH FUNCTION
136
DOES LANGUAGE IDENTIFIER MATCH IMPLEMENTATION LANGUAGE?
YES
138
DISPATCHER RETURNS NATIVE LANGUAGE REFERENCE
NO
140
DISPATCHER QUERIES SOMRef FOR EXISTING IMPLEMENTATION LANGUAGE PROXY
142
APPROPRIATE PROXY LOCATED?
NO
144
DISPATCHER ALLOCATES NEW PROXY
YES
146
DISPATCHER INVOKES METHOD ON IMPLEMENTATION OBJECT
148
ANY RETURN VALUE OR EXCEPTION?
YES
150
DISPATCHER CONVERTS IMPLEMENTATION AND PROXY REFERENCES TO SOMRefs
NO
END?
152
PROXY CONVERTS SOMRefs BACK TO IMPLEMENTATION AND PROXY REFERENCES

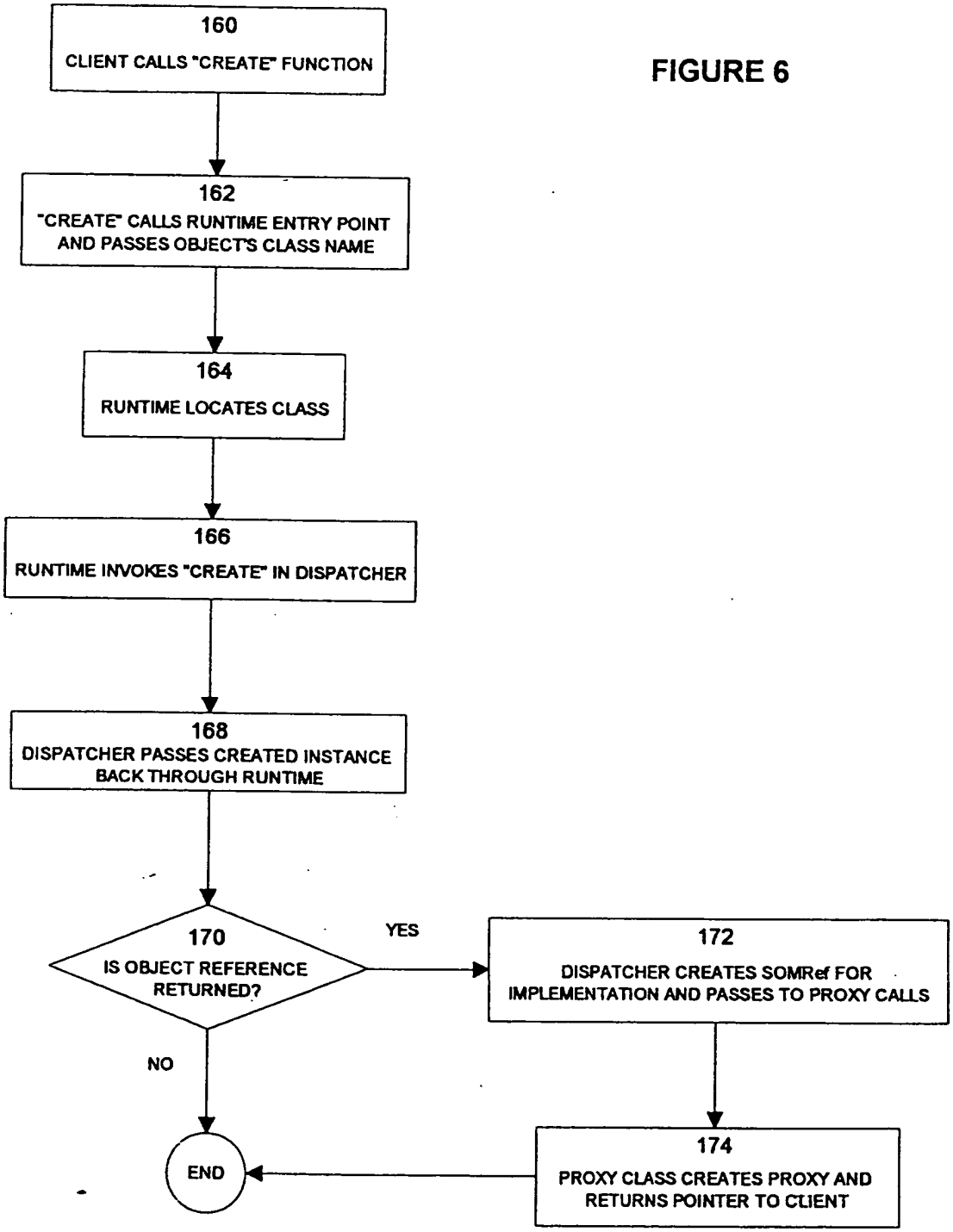

160
CLIENT CALLS "CREATE" FUNCTION
162
"CREATE" CALLS RUNTIME ENTRY POINT AND PASSES OBJECT'S CLASS NAME
164
RUNTIME LOCATES CLASS
166
RUNTIME INVOKES "CREATE" IN DISPATCHER
168
DISPATCHER PASSES CREATED INSTANCE BACK THROUGH RUNTIME
170
IS OBJECT REFERENCE RETURNED?
YES
172
DISPATCHER CREATES SOMRef FOR IMPLEMENTATION AND PASSES TO PROXY CALLS
NO
END
174
PROXY CLASS CREATES PROXY AND RETURNS POINTER TO CLIENT

FIGURE 6